

Priority-based task reassignments in hierarchical 2D mesh-connected systems using tableaux

Dohan Kim

Abstract

Task reassignments in 2D mesh-connected systems (2D-MSs) have been researched and simulated for several decades. We propose a hierarchical 2D mesh-connected system (2D-HMS) in order to exploit the regular nature of a 2D-MS. In our approach priority-based task assignments and reassignments in a 2D-HMS are represented by tableaux and their algorithms. We provide examples of priority-based task reassignments in a 2D-HMS in which task relocations are simply reduced to a jeu de taquin slide.

Keywords: Task relocation, Task reassignment, Young tableau, 2D mesh, Jeu de taquin

1. Introduction

A distributed system is a collection of processors connected by an arbitrary interconnection network [1, 2]. Among various interconnection networks for distributed systems, two-dimensional (2D) mesh has received extensive study due to its simplicity, efficiency, and structural regularity [3–6]. Some data structures, such as matrices and arrays, naturally fit into a 2D mesh-connected system (2D-MS) [6]. Since tasks are often assigned to a submesh in a 2D-MS, continuous submesh allocations and deallocations of different sizes may cause *fragmentation* [3, 4] in a 2D-MS. Task relocation is an approach to decrease fragmentation by reassigning a running task to an idle processor in a 2D-MS, which involves capturing and transferring the state of the running task to the idle processor [3, 7]. Meanwhile, the main objectives of task assignments and reassignments in distributed systems are their applications to high performance computing [7, 8]. In a similar vein our

Email address: dohankim1@gmail.com (Dohan Kim)

objective of task reassignments in a 2D-MS is to minimize task turnaround time rather than reducing fragmentation.

For decades, a wide variety of ways to tackle task assignment and re-assignment problems have been researched, such as graph-theoretic [9–14], mathematical programming [15], and heuristics [16, 17]. One of the common methods to represent and solve a task assignment problem is a graph-theoretic method, which uses a graph-matching algorithm between a task graph and a processor graph [9–11]. To complement the graph-theoretic method, our previous work [18] presented the *Young tableaux* [19–22] approach to representing task assignments. In this paper we convert a 2D-MS into a 2D mesh processor graph. Then, a 2D mesh processor graph is represented by a *Young diagram* [19, 21], and its task assignment is represented by a tableau. Our greedy task relocation policy is based on a hierarchical 2D-MS in which task relocations are performed systematically by using tableau algorithms.

The remainder of this paper is organized as follows. Section 2 presents the problem statement along with its assumptions. We provide an introduction to tableaux and their algorithms in Section 3. In Section 4 we present how 2D mesh graphs are converted into tableaux. Section 5 shows how task relocations under the greedy task relocation policy in a 2D-HMS are reduced to a jeu de taquin slide, and how they are applied in a 2D-HMS. Section 6 gives an application of our approach to priority-based task reassignments in a 2D-HMS. Finally, we conclude in Section 7.

2. Priority-based task reassignment problem in a 2D-HMS

We first give the necessary definitions of task assignments in a heterogeneous system. Then, we present our priority-based task reassignment problem in a hierarchical 2D mesh-connected system along with its assumptions.

Definition 2.1 ([11]). A *task graph* $T = (V, E)$ is a directed acyclic graph, where each vertex in $V = \{v_1, v_2, \dots, v_n\}$ represents a task, and each edge $(v_i, v_j) \in E \subset V \times V$ represents the precedence relationships between tasks, i.e., v_j cannot begin its execution before v_i completes its execution.

Definition 2.2 ([23–25]). A heterogeneous system P is a set of m heterogeneous processors $P = \{p_1, p_2, \dots, p_m\}$ whose communications are described by a network topology. If $T = (V, E)$ is a task graph, then the execution time of $v_i \in V$ on $p_j \in P$ is the function $\omega : V \times P \rightarrow \mathbb{Q}^+$. A heterogeneous

system P is said to be *consistent* if processor $p_x \in P$ executes a task n times faster than processor $p_y \in P$, then it executes all other tasks n times faster than processor p_y .

Let $T = (V, E)$ be a task graph and $P = \{p_1, p_2, \dots, p_m\}$ be a heterogeneous system. A startup cost of initiating a task on a processor is assumed to be negligible. In a consistent system, the computation cost of task v_s on p_t is defined by $\omega(v_s, p_t) = r(v_s)/c(p_t)$, where $r(v_s)$ is the computation or resource requirement of task v_s , and $c(p_t)$ is the execution rate or capacity of processor p_t [24, 26]. In an inconsistent system, which is a generalization of a consistent system, the computation cost of task v_s on p_t is defined by $\omega(v_s, p_t) = w_{st}$ in which w_{st} is the corresponding $(s, t)^{\text{th}}$ entry in a $|V| \times |P|$ cost matrix W [24]. In the remainder of this paper, we assume that every 2D-MS is consistent.

Definition 2.3 ([27, 28]). A 2D mesh-connected system consists of $m \times n$ processors structured as a rectangular grid of height m and width n . Each processor is addressed by its coordinate (i, j) for $1 \leq i \leq m$ and $1 \leq j \leq n$. An internal processor (x, y) , where $1 < x < m$ and $1 < y < n$, is directly connected to its four neighboring processors $(x - 1, y)$, $(x + 1, y)$, $(x, y - 1)$, $(x, y + 1)$. A processor in the four corners has two neighboring processors, while a processor in the remaining boundary has three neighboring processors, respectively (see Fig. 3(d) in Section 4).

Definition 2.4. A *hierarchical 2D mesh-connected system* of $m \times n$ heterogeneous processors is a heterogeneous 2D mesh-connected system with the following partial order:

$$P(i - 1, j) \prec_p P(i, j), \quad P(i, j - 1) \prec_p P(i, j), \quad 1 < i \leq m, \quad 1 < j \leq n,$$

where $P(a, b) \prec_p P(c, d)$ means that a processor addressed by (a, b) has a higher priority (or execution rate) than a processor addressed by (c, d) .

Definition 2.5 ([9–11]). Let $T_m = \{t_1, t_2, \dots, t_m\}$ be a set of m tasks with or without precedence constraints and $R_n = \{p_1, p_2, \dots, p_n\}$ be a set of n processors. Let A be a task assignment function between T_m and R_n . Let $t_p^e(A)$ denote the total execution time of processor p for the task assignment A and let $t_p^i(A)$ denote the total idle time of processor p for the task assignment A . The *turnaround time* of processor p for the task assignment A is the total time spent in the processor p for the task assignment A . Let $t_p(A) \stackrel{\text{def}}{=} t_p^e(A) + t_p^i(A)$.

$t_p^i(A) + t_p^e(A)$ and $t(A) \stackrel{\text{def}}{=} \max_p t_p(A)$. We call $t(A)$ the task turnaround time of the task assignment A . An *optimal task assignment* is the task assignment A' that minimize the task turnaround time:

$$t(A') = \min_A t(A) = \min_A \max_p t_p(A) .$$

We next define the necessary conditions for being an optimal task assignment in a 2D-HMS.

Definition 2.6. Let $T_m = \{1, 2, \dots, m\}$ be a set of m tasks having priorities represented by task IDs from 1 to m , where a lower task ID indicates a higher priority¹. A total order relation \prec_t is defined on T_m as follows. $t_1 \prec_t t_2$ for any two tasks $t_1 \in T_m$ and $t_2 \in T_m$ means that t_1 has a higher priority than t_2 . Let R_n be a 2D-HMS consist of n ($n \geq m$) heterogeneous processors whose priorities (or execution rates) of rows and columns are sorted in descending order. Let A be a bijective, priority-based task assignment between T_m and m processors in R_n . The necessary conditions for being an optimal task assignment between T_m and m processors in R_n are defined as:

- (i) Priorities of the assigned tasks on processors strictly decrease in rows and columns, i.e., $t_1 \prec_t t_2$ whenever $A(t_1) \prec_p A(t_2)$ for any two tasks $t_1 \in T_m$ and $t_2 \in T_m$.
- (ii) Processors of $A(T_m)$ in R_n is left-justified, where the row sizes of $A(T_m)$ are weakly decreasing.

The first condition in Definition 2.6 ensures that a processor with a higher priority executes a task with a higher priority. The second condition in Definition 2.6 ensures that if one processor is idle and the other processor is busy for two adjacent processors in a 2D-HMS, the processor with the lower priority is chosen to be idle. We say that a task assignment or reassignment A in a 2D-HMS is *necessarily optimal* if it satisfies the necessary conditions for being an optimal task assignment in a 2D-HMS. However, these conditions are not universal for task assignments in 2D-MSs in that they depend on priority schemes and system characteristics. We focus on priority-based task assignments in a 2D-HMS in which an optimal task assignment for priority-based tasks satisfies the necessary conditions in Definition 2.6. Now we define the priority-based task reassignment problem in a 2D-HMS.

¹In some priority schemes [24, 29], a higher number indicates a higher priority. Throughout this paper, it is assumed that a lower number indicates a higher priority.

The *priority-based task reassignment problem* in a 2D-HMS is to find a necessarily optimal task reassignment sequence $(A_k)_{k=0}^{m-1}$ for the given initial task assignment $A = A_0$ along with the task completion sequence $(b_i)_{i=1}^m$, where $b_i \in T_m$ for $1 \leq i \leq m$. The constraints and assumptions that we have made are as follows:

- (1) Both tasks and processors are heterogeneous.
- (2) Each processor can process at most one task at a time. Processors are dedicated and consistent to the priority-based task assignment, where no other task is processed when the priority-based task assignment is executed.
- (3) Each priority-based task reassignment is achieved by an iterative sequence of task relocations, where each task relocation is allowed between two adjacent processors in a 2D-HMS if one processor is in idle state and the other node is in busy state.
- (4) Two task relocations do not occur simultaneously in a 2D-HMS.
- (5) Task relocation and communication costs are ignored.
- (6) The number of tasks are less than or equal to the number of the available processors in a 2D-HMS. Otherwise, a *clustering* [26] of tasks needs to be employed as an intermediate step, which is beyond the scope of this paper.

Most of the traditional approaches [3, 4, 6, 27, 28] are based on the assumptions that processors in a 2D-MS are homogeneous. Further, the topology of the assigned tasks on a 2D-MS is restricted (e.g., rectangular or square-mesh shape) and priority-based task assignments have not often been considered [3, 4]. Thus, there is a lack of systematic mechanisms of task relocations in a 2D-HMS. We use tableaux and their algorithms for the priority-based task reassignment problem in a 2D-HMS. When representing a 2D-MS or a 2D-HMS, there is also a tradeoff between using a Young diagram (or a tableau) versus a 2D mesh graph. Since a Young diagram or a tableau is intended to represent a 2D mesh graph in a compact manner, it does not efficiently express communication costs in a 2D-HMS. If we ignore communication costs, a 2D mesh graph has an equivalent form of a Young diagram or a tableau. In the next section we give an introduction to tableaux and their algorithms.

3. Tableaux and their algorithms

A tableau is one of the essential tools to study combinatorics [21, 30], representation theory [20, 22], symmetric functions [19, 31], etc. We first introduce the required definitions of tableaux and their combinatorial algorithms.

Definition 3.1 ([19]). A *partition* of n is defined as a sequence $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_i)$, where the λ_k are weakly decreasing and $\sum_{k=1}^i \lambda_k = n$. If λ is a partition of n , then we write $\lambda \vdash n$.

Definition 3.2 ([18, 19, 22]). Let $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_i) \vdash n$. A *Young diagram* (or *Ferrers diagram*) of shape λ is a left-justified, finite collection of cells, with row j containing λ_j cells for $1 \leq j \leq i$.

Definition 3.3 ([19]). Let λ be a partition. An *inner corner* of the Young diagram of shape λ is a cell $(i, j) \in \lambda$ whose removal leaves the Young diagram of a partition. An *outer corner* of the Young diagram of shape λ is a cell $(i, j) \notin \lambda$ whose addition results in the Young diagram of a partition.

Definition 3.4 ([19, 22, 32]). Let $\lambda \vdash n$. A *tableau* t of shape λ is a Young diagram of shape λ filled with a set of elements, often positive integers. A *Young tableau* T of shape λ is a tableau of shape λ whose entries are the numbers from 1 to n , each occurring once.

Definition 3.5 ([22]). A *standard Young tableau* is a Young tableau whose entries are strictly increasing in rows and columns.

Definition 3.6 ([19]). Let $\lambda \vdash n$. A *partial tableau* p of shape λ is a tableau whose entries are strictly increasing in rows and columns.

Note that a partial tableau p in Definition 3.6 is the standard Young tableau if the entries of p are exactly $\{1, 2, \dots, n\}$.

Example. Consider $t_1 = \begin{array}{|c|c|c|} \hline 1 & 3 & 5 \\ \hline 2 & 4 & \\ \hline 6 & & \\ \hline \end{array}$, $t_2 = \begin{array}{|c|c|c|} \hline 1 & 3 & 5 \\ \hline 4 & 2 & \\ \hline 6 & & \\ \hline \end{array}$. We see that t_1 is standard, but t_2 is not.

The number of standard Young tableaux of the given shape λ is obtained from the *hook formula*.

Definition 3.7 ([20, 22]). If $\nu = (i, j)$ is a cell in the Young diagram of shape λ , then the *hook* of ν , denoted H_ν , is the set of all cells directly to the right of ν or directly below ν , including ν itself:

$$H_\nu = H_{i,j} = \{(i, j') : j' \geq j\} \cup \{(i', j) : i' \geq i\}.$$

The *hook length* of $\nu = (i, j)$, denoted $h_{i,j}$, is the number of cells in its hook, i.e., $h_{i,j} = |H_{i,j}|$.

Theorem 3.1 ([33]). If $\lambda \vdash n$, then the number f^λ of standard Young tableaux of shape λ is

$$f^\lambda = \frac{n!}{\prod_{(i,j) \in \lambda} h_{i,j}}. \quad \square$$

Example. Labeling each cell with its hook length for the Young diagram of shape $(3, 2, 1)$ and $(4, 4, 4, 4)$ are given below:

$$\begin{array}{|c|c|c|} \hline 5 & 3 & 1 \\ \hline 3 & 1 & \\ \hline 1 & & \\ \hline \end{array}, \quad \begin{array}{|c|c|c|c|} \hline 7 & 6 & 5 & 4 \\ \hline 6 & 5 & 4 & 3 \\ \hline 5 & 4 & 3 & 2 \\ \hline 4 & 3 & 2 & 1 \\ \hline \end{array}.$$

If the shape is $\lambda = (3, 2, 1)$, then $f^{(3,2,1)} = 6!/(5 \cdot 3^2 \cdot 1^3) = 16$. If the shape is $\lambda = (4, 4, 4, 4)$, then $f^{(4,4,4,4)} = 16!/(7 \cdot 6^2 \cdot 5^3 \cdot 4^4 \cdot 3^3 \cdot 2^2 \cdot 1) = 24024$.

Definition 3.8 ([19, 21]). Let $\mu = (\mu_1, \mu_2, \dots, \mu_i)$ and $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_j)$ be partitions with $\mu \subseteq \lambda$ (i.e., $i \leq j$ and $\mu_k \leq \lambda_k$ for $1 \leq k \leq i$). Then a *skew shape* of λ/μ is the set of cells

$$\lambda/\mu = \{c : c \in \lambda \text{ and } c \notin \mu\}.$$

A skew shape of λ/μ is *normal* if $\mu = \emptyset$. A tableau of skew shape λ/μ is called a *skew tableau* of shape λ/μ . A *partial skew tableau* of shape λ/μ (or a partial tableau of skew shape λ/μ) is a skew tableau of shape λ/μ whose entries are strictly increasing in rows and columns. A partial skew tableau is called the *standard skew tableau* if its entries are precisely $\{1, 2, \dots, n\}$.

Example. Consider skew tableaux of shape $\lambda/\mu = (4, 3, 3, 2)/(2, 2)$ given by

$$t_1 = \begin{array}{|c|c|c|} \hline 1 & 7 & \\ \hline 3 & 5 & \\ \hline 2 & 4 & 5 \\ \hline 6 & 9 & \\ \hline \end{array}, \quad t_2 = \begin{array}{|c|c|c|} \hline 1 & 7 & \\ \hline 5 & 3 & \\ \hline 2 & 4 & 3 \\ \hline 6 & 9 & \\ \hline \end{array}.$$

We see that t_1 is a partial skew tableau, but t_2 is not.

Definition 3.9 ([34]). A group (G, \cdot) is a nonempty set G , closed under a binary operation \cdot , such that the following axioms are satisfied:

- (i) $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in G$,
- (ii) There is an element $e \in G$ such that for all $x \in G$, $e \cdot x = x \cdot e = x$,
- (iii) For each element $a \in G$, there is an element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

Definition 3.10 ([35]). The group of all bijections $I_n \rightarrow I_n$, where $I_n = \{1, 2, \dots, n\}$, is called the *symmetric group on n letters*, denoted as \mathfrak{S}_n .

Since \mathfrak{S}_n is the group of all permutations of $I_n = \{1, 2, \dots, n\}$, the order of \mathfrak{S}_n is $n!$ [34].

Definition 3.11 ([34, 35]). Let i_1, i_2, \dots, i_n be distinct elements of $I_n = \{1, 2, \dots, n\}$. Then $\begin{pmatrix} 1 & 2 & \cdots & n \\ i_1 & i_2 & \cdots & i_n \end{pmatrix} \stackrel{\text{def}}{=} [i_1 i_2 \cdots i_n] \in \mathfrak{S}_n$ denotes the permutation that maps $1 \mapsto i_1, 2 \mapsto i_2, \dots, n \mapsto i_n$.

Now we discuss combinatorial algorithms of tableaux, including row insertion, Robinson-Schensted, and jeu de taquin. We first describe the row insertion algorithm. The *row insertion* or *row bumping algorithm* [20] takes a partial tableau P , and a positive number x that is not in P . Then, it constructs the new partial tableau P' . The procedure of row insertion is described by Algorithm 1, which always yields a partial tableau. Fig. 1 de-

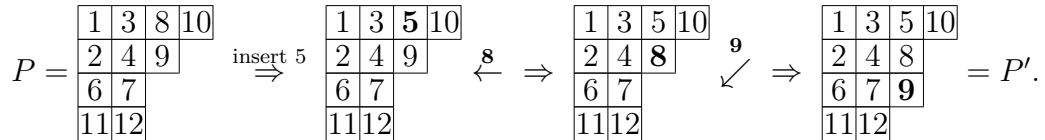


Fig. 1. An example of row insertion.

scribes the procedure of row-inserting $x = 5$ into P . Entries that are bumped during the row insertion are indicated by boldface type in Fig. 1. The row insertion procedure is always reversible. If we have P' along with the address of the added cell in P' , we can restore the original partial tableau P . The procedure of *reverse bumping* [20] is described in Algorithm 2.

Algorithm 1: Row insertion (Row bumping) [19, 20]

Input: a partial tableau P and an input number x that is not in P

Output: a partial tableau P' (x is inserted into P)

begin

 set $i := 1$ as the first row of P ;

while x is less than some element of row i **do**

 let y be the smallest element of row i that is greater than x ;

 replace y by x ; set $x := y$ and $i := i + 1$;

end

 place x (with its new cell) at the end row of i ; // x is the largest element of row i

end

Algorithm 2: Reverse bumping [20]

Input: a partial tableau P' and number $x = P'_{i,j}$ in P' (cell (i, j) has to be an outer corner in the underlying Young diagram of P').

Output: a partial tableau P

begin

 set $x' := x$ and $k := i - 1$;

 remove the cell of x (including x itself) in P' ;

while $k \neq 0$ **do**

 let y be the largest element of row k that is smaller than x' ;

 replace y by x' ; set $x' := y$ and $k := k - 1$;

end

end

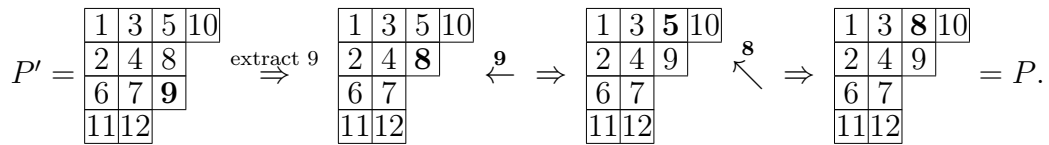


Fig. 2. An example of reverse bumping.

The address of the added cell in P' in Fig. 1 is $(3, 3)$ having an entry 9. Fig. 2 shows the procedure of reverse-bumping $x = 9$ to P' . Note that it restores the original partial tableau P in Fig. 1.

We next describe the Robinson-Schensted algorithm. The *Robinson-*

Schensted algorithm [19, 36] provides a bijection between elements of the symmetric group \mathfrak{S}_n and pairs of standard Young tableaux of the same shape $\lambda \vdash n$. Recall that $[i_1 i_2 \cdots i_n] \in \mathfrak{S}_n$ denotes the permutation that maps $1 \mapsto i_1, 2 \mapsto i_2, \dots, n \mapsto i_n$. For each $\pi = [i_1 i_2 \cdots i_n] \in \mathfrak{S}_n$, the Robinson-Schensted algorithm constructs a sequence of Young tableaux pairs $(\emptyset, \emptyset) = (P_0, Q_0), (P_1, Q_1), \dots, (P_n, Q_n) = (P, Q)$ (see Algorithm 3). By the row insertion algorithm, i_1, i_2, \dots, i_n are inserted into the P 's. Meanwhile, numbers $1, 2, \dots, n$ are simply placed sequentially to the Q 's so that the shape of P_k and the shape of Q_k are the same for $1 \leq k \leq n$ [19]. We call P the *insertion tableau* and Q the *recording tableau* of π , denoted $P(\pi)$, and $Q(\pi)$, respectively [19, 21].

Algorithm 3: Robinson-Schensted algorithm $(\pi \rightarrow (P, Q))$ [19, 36]

Input: $\pi = [x_1 x_2 \cdots x_n] \in \mathfrak{S}_n$

Output: $P = P_n$ and $Q = Q_n$ (the insertion tableau and the recording tableau of π , respectively)

begin

 set $(P_0, Q_0) := (\emptyset, \emptyset)$;

for $k \leftarrow 1$ **to** n **do**

 construct P_k by inserting x_k into P_{k-1} ; place k into Q_{k-1} so that the shape of P_k and Q_k are the same;

end

return (P_n, Q_n) ;

end

Note that the insertion tableau P and the recording tableau Q of $\pi = [i_1 i_2 \cdots i_n] \in \mathfrak{S}_n$ are both standard Young tableaux of the same shape. The procedure of Algorithm 3 is reversible, i.e., $(P, Q) \rightarrow \pi$. Given (P_k, Q_k) , we retrieve the k th element of π (i.e., i_k) from (P_k, Q_k) , and find P_{k-1} with the i_k removed by using the reverse bumping algorithm (see Algorithm 2). We easily find Q_{k-1} from Q_k by removing k (along with its cell) from Q_k . Thus, given $(P, Q) = (P_n, Q_n)$, we construct a sequence of standard Young tableaux pairs $(P_n, Q_n), (P_{n-1}, Q_{n-1}), \dots, (P_0, Q_0) = (\emptyset, \emptyset)$, where all the elements of $\pi \in \mathfrak{S}_n$ are recovered in reverse order [19]. The theorem about the Robinson-Schensted algorithm is as follows.

Theorem 3.2 ([21, 36]). *The Robinson-Schensted algorithm establishes a bijection between elements of \mathfrak{S}_n and pairs of standard Young tableaux of the*

same shape $\lambda \vdash n$. In other words, $\sum_{\lambda \vdash n} (f^\lambda)^2 = n!$, where f^λ is the number of standard Young tableaux of shape $\lambda \vdash n$. \square .

Note that the bijection in the Robinson-Schensted algorithm is given by $\pi \rightarrow (P, Q)$ rather than $\pi \rightarrow P$. Each insertion tableau P of $\pi = [x_1 x_2 \cdots x_n] \in \mathfrak{S}_n$ corresponds to an equivalence class of \mathfrak{S}_n , called a P -equivalence class.

Definition 3.12 ([19]). Two permutations $\pi = [x_1 x_2 \cdots x_n] \in \mathfrak{S}_n$ and $\tau = [y_1 y_2 \cdots y_n] \in \mathfrak{S}_n$ are called P -equivalent if $P(\pi) = P(\tau)$, denoted $\pi \cong_P \tau$.

Example. Both $[2\ 1\ 3] \in \mathfrak{S}_3$ and $[2\ 3\ 1] \in \mathfrak{S}_3$ have the same insertion tableau, i.e., $[2\ 1\ 3] \cong_P [2\ 3\ 1]$. Similarly, $[1\ 3\ 2] \in \mathfrak{S}_3$ and $[3\ 1\ 2] \in \mathfrak{S}_3$ have the same insertion tableau, i.e., $[1\ 3\ 2] \cong_P [3\ 1\ 2]$.

Now we discuss *Knuth-equivalence classes* of \mathfrak{S}_n and their relationship to P -equivalence classes of \mathfrak{S}_n .

Definition 3.13 ([21]). Suppose $x < y < z$. A *Knuth transformation* of a permutation $\pi \in \mathfrak{S}_n$ is a transformation of $\pi \in \mathfrak{S}_n$ into another permutation $\tau \in \mathfrak{S}_n$ that has one of the following forms:

- (i) $\pi = [x_1 \cdots y x z \cdots x_n] \in \mathfrak{S}_n \implies \tau = [x_1 \cdots y z x \cdots x_n] \in \mathfrak{S}_n$,
- (ii) $\pi = [x_1 \cdots y z x \cdots x_n] \in \mathfrak{S}_n \implies \tau = [x_1 \cdots y x z \cdots x_n] \in \mathfrak{S}_n$,
- (iii) $\pi = [x_1 \cdots x z y \cdots x_n] \in \mathfrak{S}_n \implies \tau = [x_1 \cdots z x y \cdots x_n] \in \mathfrak{S}_n$,
- (iv) $\pi = [x_1 \cdots z x y \cdots x_n] \in \mathfrak{S}_n \implies \tau = [x_1 \cdots x z y \cdots x_n] \in \mathfrak{S}_n$.

Two permutations $\pi, \tau \in \mathfrak{S}_n$ are called *Knuth-equivalent* if one of them can be obtained from the other by a sequence of Knuth transformations, denoted $\pi \cong_K \tau$.

Example. We see that $[2\ 1\ 3] \in \mathfrak{S}_3$ and $[2\ 3\ 1] \in \mathfrak{S}_3$ are Knuth-equivalent by (i) and (ii) in Definition 3.13, written $[2\ 1\ 3] \cong_K [2\ 3\ 1]$. Similarly, $[1\ 3\ 2] \in \mathfrak{S}_3$ and $[3\ 1\ 2] \in \mathfrak{S}_3$ are Knuth-equivalent by (iii) and (iv) in Definition 3.13, written $[1\ 3\ 2] \cong_K [3\ 1\ 2]$.

From the preceding examples, we observe that some permutations in \mathfrak{S}_3 are Knuth-equivalent if they are P -equivalent, and vice versa. More generally, P -equivalence classes and Knuth-equivalence classes coincide in \mathfrak{S}_n .

Theorem 3.3 ([21, 30]). *Permutations in \mathfrak{S}_n are Knuth-equivalent if and only if they are P -equivalent.* \square

Definition 3.14 ([19, 21]). Let t be a tableau. The *reading word* or *row word* of t , denoted $r(t)$, is the permutation of entries of t obtained by concatenating the rows of t from bottom to top, i.e., $r(t) = R_k R_{k-1} \dots R_1$, where R_1, \dots, R_k are the rows of t .

Lemma 3.1 ([21]). *If π is the reading word of a standard Young tableau T , then T is $P(\pi)$.* \square

The *jeu de taquin* (or the "teasing game") of Scützenberger [21, 37] consists of a set of rules for transforming *partial tableaux*, while some properties of partial tableaux are preserved during transformations. There are two kinds of jeu de taquin slides, which are a forward and a backward slide. A forward jeu de taquin slide is described in Algorithm 4. Note that the resulting tableau of a forward jeu de taquin slide is still a partial tableau.

Algorithm 4: A forward jeu de taquin slide [19, 37]

Input: a partial tableau P of skew shape λ/μ ; an inner corner of μ

Output: a partial tableau P'

begin

 pick x to be an inner corner of μ ;

while x is not an inner corner of λ **do**

if $x = (i, j)$ **then**

 let x' be the cell of $\min\{P_{i+1,j}, P_{i,j+1}\}$;

 // if only one of $P_{i+1,j}$ and $P_{i,j+1}$ exists, then choose that value as a minimum;

end

 slide $P_{x'}$ into cell x and set $x := x'$;

end

return the partial tableau P' ;

end

A backward jeu de taquin slide can be an invertible operation of a forward jeu de taquin slide by performing the backward slide into the cell that was vacated by the forward slide. A backward jeu de taquin slide is described in Algorithm 5.

Algorithm 5: A backward jeu de taquin slide [19, 37]

Input: a partial tableau P of skew shape λ/μ ; an outer corner of λ

Output: a partial tableau P'

begin

 pick y to be an outer corner of λ ;

while y is not an outer corner of μ **do**

if $y = (i, j)$ **then**

 let y' be the cell of $\max\{P_{i-1,j}, P_{i,j-1}\}$;

 // if only one of $P_{i-1,j}$ and $P_{i,j-1}$ exists, then choose that value as a maximum;

end

 slide $P_{y'}$ into cell y and set $y := y'$;

end

return the partial tableau P' ;

end

Example. Consider the following jeu de taquin slides for $x = (1, 1)$ and $y = (2, 4)$ in P :

$$P : \begin{array}{|c|c|c|c|} \hline & 3 & 5 & 9 \\ \hline 2 & 4 & 8 & \\ \hline 6 & 7 & 10 & \\ \hline \end{array}, \quad \text{jdt}^x(P) : \begin{array}{|c|c|c|c|} \hline 2 & 3 & 5 & 9 \\ \hline 4 & 7 & 8 & \\ \hline 6 & 10 & & \\ \hline \end{array}, \quad \text{jdt}_y(P) : \begin{array}{|c|c|c|c|} \hline & & 3 & 5 \\ \hline 2 & 4 & 8 & 9 \\ \hline 6 & 7 & 10 & \\ \hline \end{array}.$$

If we denote the forward jeu de taquin slide on P from cell $x = (1, 1)$ in P as $\text{jdt}^x(P)$, and backward jeu de taquin slide on P from cell $y = (2, 4)$ in P as $\text{jdt}_y(P)$, then both $\text{jdt}^x(P)$ and $\text{jdt}_y(P)$ are still partial tableaux. Furthermore, if we let cell $(3, 3)$ of $\text{jdt}^x(P)$ as z , we have $\text{jdt}_z(\text{jdt}^x(P)) = P$, which restores P .

Definition 3.15 ([21]). Partial tableaux P and P' are called *jeu de taquin equivalent*, written $P \cong_{\text{jdt}} P'$, if P' can be obtained from P by some sequence of jeu de taquin slides, or vice versa.

Theorem 3.4 ([19, 37]). *The jeu de taquin equivalence class of a given partial skew tableau P contains exactly one partial tableau of normal shape, denoted $j(P)$.* \square

Theorem 3.4 says that jeu de taquin slides bring a partial tableau P of skew shape to a partial tableau $j(P)$ of normal shape. Recall that a skew

shape λ/μ is a normal shape if $\mu = \emptyset$. The following results show that the jeu de taquin equivalence relation is closely connected to both the P -equivalence and the Knuth-equivalence relation.

Lemma 3.2 ([21]). *Each jeu de taquin slide converts the reading word of a standard skew tableau into a Knuth-equivalent one.* \square

Theorem 3.5 ([21, 37]). *Let P and P' be standard skew tableaux. They are jeu de taquin equivalent, i.e., $P \cong_{jdt} P'$, if and only if their reading words are Knuth-equivalent, i.e., $r(P) \cong_K r(P')$.*

Proof. $P \cong_{jdt} P' \Leftrightarrow j(P) = j(P')$ (by Theorem 3.4) $\Leftrightarrow r(j(P)) \cong_K r(j(P'))$ (by Lemma 3.1 and Theorem 3.3) $\Leftrightarrow r(P) \cong_K r(P')$ (by Lemma 3.2). \square

4. 2D mesh graphs and tableaux

Recall that $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subset V$ and $E' \subset E$ [38]. A *vertex-induced subgraph* G' is the subgraph of G that contains all edges of G that join any two vertices in V' of G' [38, 39]. In this section we present a certain type of vertex-induced subgraphs of 2D mesh graphs and their relationship to tableaux.

Definition 4.1. A *2D mesh graph* G of shape $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \vdash n$ is a left-justified, a vertex-induced subgraph of a $k \times \lambda_1$ 2D mesh graph, where each row contains λ_i vertices for $1 \leq i \leq k$. A 2D mesh graph $G = (V, E)$ of shape λ is *directed* if all edges of G are directed in such a way that the horizontal edges are directed from left to right and the vertical edges are directed from top to bottom.

Similarly to the traditional 2D mesh graphs, each vertex of 2D mesh graph $G = (V, E)$ of shape $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \vdash n$ is represented by its coordinate. For example, a top-left corner vertex is $(1, 1)$ while a bottom-right corner vertex is (k, λ_k) . If $\lambda_1 = \lambda_2 = \dots = \lambda_k$ for shape $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \vdash n$, we say that λ is of *canonical shape*.

Definition 4.2. Let $G_1 = (V_1, E_1)$ be a 2D mesh graph of shape λ and $G_2 = (V_2, E_2)$ be a 2D mesh graph of shape μ , where $\mu = (\mu_1, \mu_2, \dots, \mu_i) \subseteq \lambda = (\lambda_1, \lambda_2, \dots, \lambda_j)$, i.e., $i \leq j$ and $\mu_k \leq \lambda_k$ for $1 \leq k \leq i$, respectively. A 2D mesh graph G' of skew shape λ/μ is the vertex-induced subgraph of G_1 , where the vertex set of G' consists of $V_1 - V_2$.

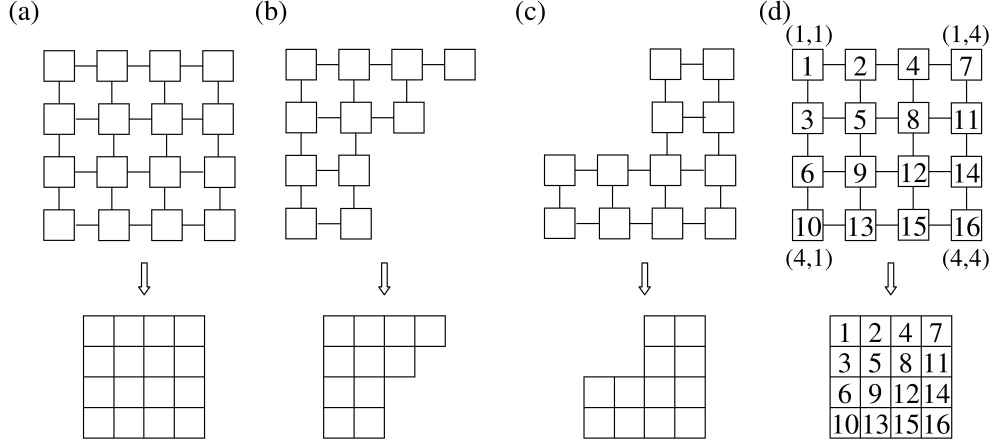


Fig. 3. Conversion of a 2D mesh graph into a Young diagram or a Young tableau.

For example, the upper figures in Fig. 3(a), (b), and (c) show 2D mesh graphs of shape $\lambda_1 = (4, 4, 4, 4)$, $\lambda_2 = (4, 3, 2, 2)$, and $\lambda_3 = \lambda/\mu$ where $\lambda = (4, 4, 4, 4)$ and $\mu = (2, 2)$, respectively. Now we consider the relationships between 2D mesh graphs and Young diagrams (or tableaux). For example, if we do not take communication costs into account for a 2D mesh processor graph, it has the compact form represented by a Young diagram or a tableau. In order to represent a 2D mesh processor or task graph of shape λ in a compact manner, we define a 2D mesh Young diagram and a 2D mesh tableau.

Definition 4.3. A *2D mesh Young diagram of shape* $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \vdash n$ is a Young diagram of shape $\lambda \vdash n$, where each cell (i, j) corresponds to each vertex (i, j) in the 2D mesh graph $G = (V, E)$ of shape $\lambda \vdash n$.

Definition 4.4. A *2D mesh Young diagram of skew shape* λ/μ is a skew Young diagram of shape λ/μ for $\mu \subseteq \lambda$. Similarly to Definition 4.3, each cell (i, j) corresponds to each vertex (i, j) in the 2D mesh graph $G = (V, E)$ of skew shape λ/μ .

Fig. 3 shows how 2D mesh graphs are converted into 2D mesh Young diagrams. Fig. 3(a) shows a 2D mesh Young diagram of canonical shape while Fig. 3(c) shows a 2D mesh Young diagram of skew shape. We next define a 2D mesh tableau in order to represent a labeled 2D mesh graph (see Fig. 3(d)).

Definition 4.5. A *2D mesh tableau of shape* $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \vdash n$ is a Young tableau of shape $\lambda \vdash n$, denoted as $(MT_{i,j})$ of shape λ or $\lambda\text{-(}MT_{i,j}\text{)}$, where each cell has a unique number from 1 to n and its underlying Young diagram is a 2D mesh Young diagram of shape $\lambda \vdash n$.

Definition 4.6. A *2D mesh tableau of skew shape* λ/μ is a tableau of skew shape λ/μ , denoted as $(MT_{i,j})$ of shape λ/μ , where each cell has a unique number from the set $\{1, 2, \dots, n\}$ and its underlying Young diagram is 2D mesh Young diagram of skew shape λ/μ .

A graph-theoretic approach to Young diagrams or tableaux has already been researched [40]. It focuses on graphs having the shape of a Young diagram or a tableau, while this paper focuses on converting a 2D mesh graph into a Young diagram or a tableau. The relationship between a certain labeling of directed acyclic graphs and the number of standard Young tableaux has been discussed in [41, 42]. The following propositions involve in counting aspects of representing 2D mesh task and processor graphs, respectively.

Proposition 4.1. *Let $G = (V, E)$ be a directed 2D mesh graph of shape $\lambda \vdash n$. Let $T = \{1, 2, \dots, n\}$ be a totally ordered set of n tasks ordered by the "less than" relation $<$. Suppose we assign tasks T to vertices V bijectively such that task ID of vertex x is less than task ID of vertex y for each edge $(x, y) \in E$. Then, the number of such assignments is the number of standard Young tableaux of shape $\lambda \vdash n$. \square*

Proposition 4.2. *Let $G = (V, E)$ be an undirected 2D mesh graph of shape $\lambda \vdash n$. Let $P = \{1, 2, \dots, n\}$ be a totally ordered set of n processors ordered by the "less than" relation $<$. Suppose we assign processors P to vertices V bijectively such that processor IDs of vertices are increasing in rows and columns. Then, the number of such assignments is the number of standard Young tableaux of shape $\lambda \vdash n$. \square*

Proposition 4.1 and 4.2 directly follow from the definition of a standard Young tableau of shape $\lambda \vdash n$. Each labeled 2D mesh task and processor graph of shape $\lambda \vdash n$ corresponds to each 2D mesh tableau of shape $\lambda \vdash n$. We call a 2D mesh task graph of shape $\lambda \vdash n$ satisfying the hypothesis of Proposition 4.1 a *standard 2D mesh task graph of shape $\lambda \vdash n$* , and a processor graph of shape $\lambda \vdash n$ satisfying the hypothesis of Proposition 4.2 a *standard 2D mesh processor graph of shape $\lambda \vdash n$* , respectively. Thus, counting the number of standard 2D mesh task or processor graphs of shape $\lambda \vdash n$ are

reduced to counting the number of 2D mesh tableaux of shape $\lambda \vdash n$ whose entries in rows and columns strictly increase, i.e., f^λ (see Theorem 3.1).

Now consider a task assignment between a standard 2D mesh task graph of shape $\lambda \vdash n$ and a standard 2D mesh processor graph of the same shape by their coordinates, i.e., each task addressed by (i, j) of the 2D mesh task graph is assigned to each processor addressed by (i, j) of the 2D mesh processor graph. Proposition 4.3 discusses the counting aspects of such task assignments.

Proposition 4.3. *Let $\lambda \vdash n$. The number of bijective task assignments by their coordinates between standard 2D mesh task graphs of shape $\lambda \vdash n$ and standard 2D mesh processor graphs of shape $\lambda \vdash n$ is $(f^\lambda)^2$. Moreover, if we sum all the numbers of such task assignments for all possible partitions (or shapes) of n , then the total number is $\sum_{\lambda \vdash n} (f^\lambda)^2 = n!$.*

Proof. By Proposition 4.1 and 4.2, the number of standard 2D mesh task and processor graphs of shape $\lambda \vdash n$ is both f^λ . Thus, for a given shape $\lambda \vdash n$, the number of bijective task assignments by their coordinates between standard 2D mesh task graphs of shape $\lambda \vdash n$ and standard 2D mesh processor graphs of shape $\lambda \vdash n$ is $(f^\lambda)^2$. For a given n , if we consider all possible partitions (or shapes) of n , then the total number of such task assignments is $\sum_{\lambda \vdash n} (f^\lambda)^2$, which is $n!$ by Theorem 3.2. \square

In case a 2D mesh graph represents a 2D mesh processor graph, we say that the associated 2D mesh Young diagram (respectively, 2D mesh tableau) is the *2D mesh processor diagram* (respectively, *2D mesh processor tableau*).

Definition 4.7. A *2D mesh processor diagram* of shape $\lambda \vdash n$ is a 2D mesh Young diagram of shape $\lambda \vdash n$, where each cell in the 2D mesh processor diagram represents each processor in the heterogeneous system $P = \{p_1, p_2, \dots, p_n\}$. Each cell is assumed to have direct communication links with its neighboring cells and have a routing capability, allowing it to send and receive messages to and from any other cell in the diagram.

To exploit the regular nature of a 2D mesh topology, we consider a hierarchical 2D mesh processor diagram of canonical shape, where the rows and columns of heterogeneous processors are sorted in descending order by their priorities (or execution rates).

Definition 4.8. A hierarchical 2D mesh processor diagram of canonical shape $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k)$ for $\lambda_1 = \lambda_2 = \dots = \lambda_k$ is a 2D mesh processor diagram consists of $k \times \lambda_1$ processors with the following partial order:

$$P(i-1, j) \prec_p P(i, j), \quad P(i, j-1) \prec_p P(i, j), \quad 1 < i \leq k, \quad 1 < j \leq \lambda_1,$$

where $P(a, b) \prec_p P(c, d)$ means that a processor addressed by (a, b) has a higher priority (or execution rate) than a processor addressed by (c, d) .

Proposition 4.4. Let $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_k) \vdash n$ for $\lambda_1 = \lambda_2 = \dots = \lambda_k$, and let $P = \{1, 2, \dots, n\}$ be a totally ordered set of $k \times \lambda_1$ processors ordered by the "less than" relation $<$. Each processor ID represents a processor priority, where $u < v$ for any two processors $u \in P$ and $v \in P$ implies that processor u has a higher priority (or execution rate) than processor v . Then, the number of ways to organize a hierarchical 2D mesh processor diagram of shape $\lambda \vdash n$ by arranging n processors in P is f^λ .

Proof. It immediately follows from Definition 3.5 and 4.8. \square

We define a *hierarchical 2D mesh tableau* in order to represent a priority-based task assignment and its reassignments in a 2D-HMS.

Definition 4.9. A *hierarchical 2D mesh tableau* of shape $\lambda \vdash n$ is a tableau of shape λ , denoted as $(\text{HMT}_{i,j})$ of shape λ or $\lambda\text{-(HMT}_{i,j})$, whose underlying 2D mesh Young diagram is a hierarchical 2D mesh processor diagram of canonical shape $\lambda \vdash n$. Entries of $\lambda\text{-(HMT}_{i,j})$ are task IDs from the set $\{1, 2, \dots, n\}$ with the total order relation \prec_t (see Definition 2.6). Empty entry of a cell is allowed in $\lambda\text{-(HMT}_{i,j})$, while all non-empty entries along with their cells must form a tableau of normal or skew shape, called a *maximally embedded tableau* of $\lambda\text{-(HMT}_{i,j})$.

Definition 4.10. Let $\lambda\text{-(HMT}_{i,j})$ be a hierarchical 2D mesh tableau of shape $\lambda \vdash n$. If the maximally embedded tableau of $\lambda\text{-(HMT}_{i,j})$ is a tableau of normal shape, we say that $\lambda\text{-(HMT}_{i,j})$ is of *normal shape*. Meanwhile, if the maximally embedded tableau of $\lambda\text{-(HMT}_{i,j})$ is a tableau of skew shape, we say that $\lambda\text{-(HMT}_{i,j})$ is of *skew shape*. If the maximally embedded tableau of $\lambda\text{-(HMT}_{i,j})$ is a partial tableau (i.e., a tableau whose entries are strictly increasing in rows and columns), we say that $\lambda\text{-(HMT}_{i,j})$ is *standard*. Otherwise, we say that $\lambda\text{-(HMT}_{i,j})$ is *generalized*.

In this paper $\lambda\text{-(HMT}_{i,j})$ is referred to as a standard $\lambda\text{-(HMT}_{i,j})$ of normal shape unless otherwise stated. Each entry in $\lambda\text{-(HMT}_{i,j})$ denotes a task, where a lower task ID indicates a higher priority. Each cell represents each processor in the hierarchical 2D mesh processor diagram. Therefore, $\lambda\text{-(HMT}_{i,j})$ represents a priority-based task assignment in a 2D mesh-connected system. Note that a priority-based task assignment represented by $\lambda\text{-(HMT}_{i,j})$ satisfies the necessary conditions for being an optimal task assignment in a 2D-HMS (see Definition 2.6).

Definition 4.11. Let $\lambda\text{-(HMT}_{i,j})$ be a hierarchical 2D mesh tableau of shape $\lambda \vdash n$. If the processor priority $P(a,b)$ is fixed for any cell (a,b) in $\lambda\text{-(HMT}_{i,j})$, we say that $\lambda\text{-(HMT}_{i,j})$ is of the fixed processor priority.

Although the priorities of underlying processors in $\lambda\text{-(HMT}_{i,j})$ have to decrease in rows and columns by Definition 4.8, it is not required to specify the processor priority for each cell (or processor) in $\lambda\text{-(HMT}_{i,j})$. Meanwhile, $\lambda\text{-(HMT}_{i,j})$ of the fixed processor priority (see Fig. 7(a)) denotes the priorities of underlying processors in $\lambda\text{-(HMT}_{i,j})$, where the label in the top-right corner of each cell denotes the processor priority (cf. [18]).

5. Priority-based task reassignments in a 2D-HMS

Consider the priority-based task assignment in Fig. 4(a) represented by $\lambda\text{-(HMT}_{i,j})$ of shape $\lambda = (3,3,3) \vdash n$ for $n = 9$. Assume $\lambda\text{-(HMT}_{i,j})$ is not of the fixed processor priority.

$$\begin{aligned}
\text{(a) } A_0 &= \begin{array}{|c|c|c|} \hline 1 & 2 & 4 \\ \hline 3 & 5 & 7 \\ \hline 6 & 8 & 9 \\ \hline \end{array}, \text{ (b) } \begin{array}{|c|c|c|} \hline \bullet & 2 & 4 \\ \hline 3 & 5 & 7 \\ \hline 6 & 8 & 9 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|} \hline 2 & \bullet & 4 \\ \hline 3 & 5 & 7 \\ \hline 6 & 8 & 9 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|} \hline 2 & 4 & \bullet \\ \hline 3 & 5 & 7 \\ \hline 6 & 8 & 9 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|} \hline 2 & 4 & 7 \\ \hline 3 & 5 & \bullet \\ \hline 6 & 8 & 9 \\ \hline \end{array} \Rightarrow \begin{array}{|c|c|c|} \hline 2 & 4 & 7 \\ \hline 3 & 5 & 9 \\ \hline 6 & 8 & \bullet \\ \hline \end{array}. \\
\text{(c) } A_0 &= \begin{array}{|c|c|c|} \hline 1 & 2 & 4 \\ \hline 3 & 5 & 7 \\ \hline 6 & 8 & 9 \\ \hline \end{array}, A_1 = \begin{array}{|c|c|c|} \hline 2 & 4 & 7 \\ \hline 3 & 5 & 9 \\ \hline 6 & 8 & \\ \hline \end{array}, A_2 = \begin{array}{|c|c|c|} \hline 2 & 4 & 7 \\ \hline 5 & 8 & 9 \\ \hline 6 & & \\ \hline \end{array}, A_3 = \begin{array}{|c|c|c|} \hline 4 & 7 & 9 \\ \hline 5 & 8 & \\ \hline 6 & & \\ \hline \end{array}, A_4 = \begin{array}{|c|c|c|} \hline 4 & 7 & 9 \\ \hline 6 & 8 & \\ \hline & & \\ \hline \end{array}, \\
A_5 &= \begin{array}{|c|c|c|} \hline 4 & 7 & 9 \\ \hline 6 & & \\ \hline & & \\ \hline \end{array}, A_6 = \begin{array}{|c|c|c|} \hline 6 & 7 & 9 \\ \hline & & \\ \hline & & \\ \hline \end{array}, A_7 = \begin{array}{|c|c|c|} \hline 7 & 9 & \\ \hline & & \\ \hline & & \\ \hline \end{array}, A_8 = \begin{array}{|c|c|c|} \hline 9 & & \\ \hline & & \\ \hline & & \\ \hline \end{array}.
\end{aligned}$$

Fig. 4. A sequence of task reassignments for a task completion sequence $(1, 3, 2, 5, 8, 4, 6, 7, 9)$.

Algorithm 6: Task reassignments: λ -(HMT $_{i,j}$) of normal shape

Input: An initial task assignment A_0 represented by λ -(HMT $_{i,j}$) of normal shape; a task completion sequence (b_1, b_2, \dots, b_m) , where $m \geq 2$, in runtime

Output: A task reassignment sequence $(A_k)_{k=0}^{m-1}$

begin

set λ -(HMT $_{i,j}^{(1)} := \lambda$ -(HMT $_{i,j}$);

for $k \leftarrow 1$ **to** $m - 1$ **do**

let t be the maximally embedded tableau of λ -(HMT $_{i,j}^{(k)}$) and μ be the shape of t ; wait for a completion of task b_k in λ -(HMT $_{i,j}^{(k)}$); if task b_k is completed, then the cell (or processor) of task b_k in λ -(HMT $_{i,j}^{(k)}$) becomes vacated; set x as the corresponding cell in λ -(HMT $_{i,j}^{(k)}$);

while x is not an inner corner of μ **do**

if $x = (i, j)$ **then**

let x' be the cell of $\min\{\text{HMT}_{i+1,j}^{(k)}, \text{HMT}_{i,j+1}^{(k)}\}$ // if only one non-idle cell (or processor) exists in $(i + 1, j)$ and $(i, j + 1)$, then choose that cell;

end

relocate the task on cell x' into cell x and set $x := x'$;

end

set $A_k := \lambda$ -(HMT $_{i,j}^{(k)}$), where λ -(HMT $_{i,j}^{(k)}$) is of normal shape;

set λ -(HMT $_{i,j}^{(k+1)} := \lambda$ -(HMT $_{i,j}^{(k)}$);

end

return the task reassignment sequence $(A_k)_{k=0}^{m-1}$;

end

If task 1 in Fig. 4(a) is completed first, the processor addressed by $(1, 1)$ becomes idle. We mark the cell $(1, 1)$ as \bullet to show that the processor addressed by $(1, 1)$ is now in the idle state. Once a processor is in the idle state, it seeks the right and below processor to perform task relocation. Recall that task relocation is only allowed between two adjacent processors if one is in the idle state and the other is in the busy state. If a processor is in the idle state, it does not check the left and above processor to perform task reloca-

tion. It is because the underlying 2D mesh processor diagram of $\lambda\text{-(HMT}_{i,j})$ is hierarchical, it is not an optimal choice if a task is to run on a processor with the lower execution rate. Thus, a processor in the idle state always seeks both the right and below processor in order to compare task priorities and to relocate a task. We see that task $\text{HMT}_{1,2}$ has a higher priority than task $\text{HMT}_{2,1}$, i.e., $2 \prec_t 3$. Thus, task relocation involves the processor addressed by $(1, 1)$ and the processor addressed by $(1, 2)$. Now task 2 has been relocated and the processor addressed by $(1, 2)$ becomes idle. If a processor becomes idle, the choice for task relocation between the right and below processor is always *greedy* [43], allowing the task with the higher priority to occupy the idle processor (see Fig. 4(b)). This process continues until no task relocation is possible, which means that the right and below processor are both idle or both not available. The final state of Fig. 4(b) is the task reassignment A_1 for the completion of task 1. Given the initial task assignment A_0 in Fig. 4(a), Fig. 4(c) shows the task reassignment sequence $(A_k)_{k=0}^{m-1}$ for $m = 9$ corresponding to the task completion sequence $(1, 3, 2, 5, 8, 4, 6, 7, 9)$. Algorithm 6 describes the procedure in Fig. 4. Task reassignments take place in Algorithm 6 if a task completion sequence is provided in run time. It turns out that the iterative greedy task relocation mechanism in Algorithm 6 corresponds to the forward jeu de taquin slide discussed in Algorithm 4, except that task relocation starts with the cell that is indicated by the task completion sequence. Note that each task assignment in a task reassignment sequence $(A_k)_{k=0}^{m-1}$ in Algorithm 6 satisfies the necessary conditions in Definition 2.6. We see that each task assignment in $(A_k)_{k=0}^{m-1}$ in Algorithm 6 is represented by a hierarchical 2D mesh tableau of normal shape, where task IDs are increasing in rows and columns on the underlying hierarchical 2D mesh processor diagram. Therefore, $(A_k)_{k=0}^{m-1}$ in Algorithm 6 is a necessarily optimal task reassignment sequence.

Thus far, we have examined the case where the initial task assignment is represented by $\lambda\text{-(HMT}_{i,j})$ of normal shape. Now we consider the case where the initial task assignment is represented by $\lambda\text{-(HMT}_{i,j})$ of skew shape.

$$t_1 = \begin{array}{|c|c|c|c|} \hline & & 1 & 6 \\ \hline & & 4 & \\ \hline 2 & 3 & 5 & \\ \hline 7 & 8 & & \\ \hline \end{array}, \quad t_2 = \begin{array}{|c|c|c|c|} \hline & & 1 & 6 \\ \hline & 3 & 4 & \\ \hline 2 & 5 & & \\ \hline 7 & 8 & & \\ \hline \end{array}, \quad t_3 = \begin{array}{|c|c|c|} \hline 1 & 6 & \\ \hline & 4 & \\ \hline 2 & 3 & 5 \\ \hline 7 & 8 & \\ \hline \end{array}, \quad t_4 = \begin{array}{|c|c|c|} \hline & 1 & 6 \\ \hline & 3 & 4 \\ \hline 2 & 5 & \\ \hline 7 & 8 & \\ \hline \end{array}, \quad t_5 = \begin{array}{|c|c|c|c|} \hline 1 & 3 & 4 & 6 \\ \hline 2 & 8 & & \\ \hline 5 & & & \\ \hline 7 & & & \\ \hline \end{array}.$$

Fig. 5. Task reassignments by using forward jeu de taquin slides.

Algorithm 7: Task reassignments: λ -(HMT $_{i,j}$) of skew shape

Input: An initial task assignment A_0 represented by λ -(HMT $_{i,j}$) of skew shape

Output: A task reassignment sequence $(A_k)_{k=0}^m$

begin

if β is a partition of m' for the maximally embedded tableau of λ -(HMT $_{i,j}$) of skew shape α/β , then set m as the value of m' ; set λ -(HMT $_{i,j}^{(1)}$) := λ -(HMT $_{i,j}$); set $k := 1$;

while the maximally embedded tableau of λ -(HMT $_{i,j}^{(k)}$) is not of normal shape **do**

if the maximally embedded tableau of λ -(HMT $_{i,j}^{(k)}$) is of (skew) shape μ/ν , pick x to be an inner corner of ν ;

while x is not an inner corner of μ **do**

if $x = (i, j)$ **then**

let x' be the cell of $\min\{\text{HMT}_{i+1,j}^{(k)}, \text{HMT}_{i,j+1}^{(k)}\}$ // if only one non-idle cell (or processor) exists in $(i+1, j)$ and $(i, j+1)$, then choose that cell;

end

relocate the task on cell x' into cell x and set $x := x'$;

end

set $A_k := \lambda$ -(HMT $_{i,j}^{(k)}$); set λ -(HMT $_{i,j}^{(k+1)}$) := λ -(HMT $_{i,j}^{(k)}$);

$k := k + 1$;

end

return the task reassignment sequence $(A_k)_{k=0}^m$, where A_m is the task reassignment represented by λ -(HMT $_{i,j}^{(m)}$) of normal shape;

end

Consider a top-left corner of a hierarchical 2D mesh tableau t_1 or t_2 in Fig. 5, where the processor with the highest priority is idle. Thus, it is a natural choice to relocate a task from a processor with the lower execution rate to a processor with the higher execution rate. Algorithm 7 describes the procedure in which the initial task assignment, represented by a hierarchical 2D mesh tableau of skew shape, is converted into the task reassignment, represented by a hierarchical 2D mesh tableau of normal shape. As shown in Fig. 5, t_3 is the maximally embedded tableau of t_1 , and t_4 is the maximally

embedded tableau of t_2 , respectively. By performing task relocations discussed in Algorithm 7 iteratively, the task reassignment t_5 is obtained from the task assignment t_1 in Fig. 5. Similarly, the task reassignment t_5 is also obtained from the task assignment t_2 in Fig. 5. If t_2 represents an initial task assignment A_0 , then t_5 corresponds to the task reassignment A_3 . According to Algorithm 7, the initial choice of task relocation for A_1 must target for either the cell $(1, 2)$ or the cell $(2, 1)$ in t_2 . Note that the task reassignment sequence (A_0, A_1, A_2, A_3) is not uniquely determined, depending on the choice of an initial task relocation. However, A_3 is uniquely determined by Theorem 3.4, since the greedy-based task relocations on $\lambda(\text{HMT}_{i,j})$ follow the forward jeu de taquin slide rules. Unlike a task reassignment represented by $\lambda(\text{HMT}_{i,j})$ of normal shape, a task reassignment represented by $\lambda(\text{HMT}_{i,j})$ of skew shape do not always satisfy the necessary conditions for being an optimal task assignment. For example, non-idle processors of A_0 , A_1 , and A_2 are not left-justified, which implies that the processor with the higher execution rate is idle for some pairs of adjacent processors in a 2D-HMS. However, the resulting task reassignment A_3 satisfies the necessary conditions for being an optimal task assignment in a 2D-HMS. Note that this process is reversible by using the backward jeu de taquin slides as discussed in Algorithm 5. Now we define an equivalence class of task reassignments up to task relocations under the greedy task relocation policy.

Definition 5.1. Two task assignments t_1 , represented by a hierarchical 2D mesh tableau $\lambda(\text{HMT}_{i,j}^1)$ of skew shape, and t_2 , represented by a hierarchical 2D mesh tableau $\lambda(\text{HMT}_{i,j}^2)$ of skew shape, are called *task reassignment equivalent* up to task relocations (under the greedy task relocation policy described in Algorithm 7), denoted as $t_1 \cong_t t_2$, if they have the same resulting task reassignment, represented by the same $\lambda(\text{HMT}_{i,j})$ of normal shape.

Proposition 5.1. Let $\lambda(\text{HMT}_{i,j}^1)$ and $\lambda(\text{HMT}_{i,j}^2)$ be hierarchical 2D mesh tableaux of skew shape whose maximally embedded tableaux are both standard skew tableaux. If two task assignments t_1 , represented by $\lambda(\text{HMT}_{i,j}^1)$, and t_2 , represented by $\lambda(\text{HMT}_{i,j}^2)$, are task reassignment equivalent, then the reading words of their maximally embedded tableaux are both Knuth-equivalent and P-equivalent.

Proof. Let P be the maximally embedded tableau of $\lambda(\text{HMT}_{i,j}^1)$ for the task assignment t_1 , and Q be the maximally embedded tableau of $\lambda(\text{HMT}_{i,j}^2)$ for the task assignment t_2 . Since $t_1 \cong_t t_2$ by hypothesis, t_1 and t_2 have the same

resulting task reassignment represented by the same $\lambda\text{-(HMT}_{i,j})$ of normal shape by Definition 5.1. Let N be the maximally embedded tableau of such $\lambda\text{-(HMT}_{i,j})$ of normal shape. Each task relocation under the greedy task relocation policy described in Algorithm 7 follows the forward jeu de taquin slide rules described in Algorithm 4. Thus, $P \cong_{jdt} Q$ by Theorem 3.4. Since P and Q are both standard skew tableaux by hypothesis satisfying $P \cong_{jdt} Q$, we conclude that the reading words of P and Q are both Knuth-equivalent and P -equivalent by Theorem 3.3 and 3.5. \square

$$\begin{aligned} \pi_k: \emptyset, \begin{array}{|c|} \hline 7 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 7 & 8 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 2 & 8 \\ \hline 7 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 7 & 8 \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 2 & 3 & 5 \\ \hline 7 & 8 & \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 2 & 3 & 4 \\ \hline 5 & 8 & \\ \hline 7 & & \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 1 & 3 & 4 \\ \hline 2 & 8 & \\ \hline 5 & & \\ \hline 7 & & \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 1 & 3 & 4 & 6 \\ \hline 2 & 8 & & \\ \hline 5 & & & \\ \hline 7 & & & \\ \hline \end{array} = \pi. \\ \\ \tau_k: \emptyset, \begin{array}{|c|} \hline 7 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 7 & 8 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 2 & 8 \\ \hline 7 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 2 & 5 \\ \hline 7 & 8 \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 2 & 3 \\ \hline 5 & 8 \\ \hline 7 & \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 2 & 3 & 4 \\ \hline 5 & 8 & \\ \hline 7 & & \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 1 & 3 & 4 \\ \hline 2 & 8 & \\ \hline 5 & & \\ \hline 7 & & \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline 1 & 3 & 4 & 6 \\ \hline 2 & 8 & & \\ \hline 5 & & & \\ \hline 7 & & & \\ \hline \end{array} = \tau. \end{aligned}$$

Fig. 6. Insertion tableaux for the reading word of t_3 and the reading word of t_4 in Fig. 5.

For example, the reading word of t_3 (i.e., the maximally embedded tableau of t_1) in Fig. 5 is $\pi = [78235416] \in \mathfrak{S}_8$. Similarly, the reading word of t_4 (i.e., the maximally embedded tableau of t_2) in Fig. 5 is $\tau = [78253416] \in \mathfrak{S}_8$. By Definition 3.13(iii), we see that π and τ are Knuth-equivalent by the choice of $x = 3$, $y = 4$, and $z = 5$. Fig. 6 shows the procedure for constructing insertion tableaux for both π and τ . We also see that π and τ are P -equivalent.

6. Application

This section presents an application of priority-based task reassignments in a 2D-HMS. We first introduce the sequential task assignment problem in a 2D-HMS.

The sequential task assignment problem in a 2D-HMS is defined as follows. A set of m tasks $T_m = \{1, 2, \dots, m\}$ with the precedence relationship $1 \rightarrow 2 \rightarrow \dots \rightarrow m$ are to be assigned to a set of m heterogeneous processors in a 2D-HMS of shape $\lambda \vdash n$ ($m \leq n$) and executed sequentially without

gaps. We assume that each task is assigned to each processor in a 2D-HMS bijectively, where a 2D-HMS is dedicated for a sequential task assignment. Tasks are heterogeneous, and their priorities are assigned by their resource requirements in which the higher task priority indicates the larger resource requirement. Find a bijective task assignment between T_m and a set of m processors in the 2D-HMS of shape $\lambda \vdash n$ ($m \leq n$) in such a way that the task turnaround time is minimized.

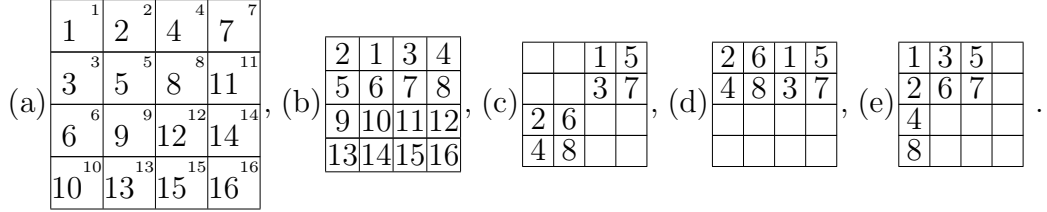


Fig. 7. Comparisons of task reassignments.

By Definition 2.6, a sequential task assignment represented by λ -(HMT $_{i,j}$) of normal shape satisfies the necessary conditions for being an optimal task assignment in a 2D-HMS. Further, an optimal task assignment is represented by λ -(HMT $_{i,j}$) of the fixed processor priority in which task ID and processor ID are the same for each cell (see Fig. 7(a)). Unless a sequential task assignment is represented by λ -(HMT $_{i,j}$) of the fixed processor priority, we only concern the necessary conditions for being an optimal task assignment. We define a *descent pair* of a generalized λ -(HMT $_{i,j}$), which breaks the first necessary condition for being an optimal task assignment in a 2D-HMS (see Definition 2.6).

Definition 6.1. Let λ -(HMT $_{i,j}$) be a generalized hierarchical 2D mesh tableau. Suppose HMT $_{i,j} \prec_t$ HMT $_{i-1,j}$ (respectively, HMT $_{i,j} \prec_t$ HMT $_{i,j-1}$). Then, $\{(i-1, j), (i, j)\}$ (respectively, $\{(i, j-1), (i, j)\}$) is called a *descent pair* of a generalized λ -(HMT $_{i,j}$).

If a descent pair occurs in a sequential task assignment represented by a generalized λ -(HMT $_{i,j}$), it always has the better task assignment in terms of task turnaround time. For example, swapping tasks on a descent pair reduces task turnaround time for a sequential task assignment in a 2D-HMS. It follows that if a sequential task assignment cannot be represented by (a standard) λ -(HMT $_{i,j}$), it is not a necessarily optimal task assignment. We

see that the task assignment represented by Fig. 7(b) has a descent pair in a generalized λ -(HMT $_{i,j}$), i.e., $\text{HMT}_{1,2} \prec_t \text{HMT}_{1,1}$. Now consider priority-based task reassignments for a sequential task assignment in a 2D-HMS. We see that the task completion sequence is simply $(1, 2, \dots, m)$ for a sequential task assignment of m tasks. Note that a greedy task relocation policy keeps λ -(HMT $_{i,j}$) from occurring any descent pair for a sequential task reassignment. If we assume that every 2D-HMS is consistent (see Definition 2.2) in which task relocations are cost-free, we have Proposition 6.1.

Proposition 6.1. *Let A_0 be a sequential task assignment for m ($m \geq 2$) tasks represented by λ -(HMT $_{i,j}$) of normal shape. Let T_1 be the task turnaround time for A_0 without task relocation, and T_2 be the task turnaround time with the task reassignment sequence $(A_k)_{k=0}^{m-1}$ (see Algorithm 6). Then, T_2 is less than T_1 , i.e., $T_2 < T_1$.*

Proof. It suffices to show that each task relocation allows each task to reduce its task execution time for a sequential task assignment in a 2D-HMS. Let $t_{i,x}$ be the task execution time of task i on processor (or cell) x in λ -(HMT $_{i,j}$) before task relocation. After task relocation, task i is relocated from processor x to its adjacent processor x' such that $P(x') \prec_p P(x)$, where $P(x') \prec_p P(x)$ means that an execution rate of processor x' is higher than that of processor x . Thus, $t_{i,x'} < t_{i,x}$. Since each task reassignment consists of iterative task relocations by Algorithm 6, we conclude that $T_2 < T_1$. \square

Now consider a sequential task assignment in a 2D-HMS represented by λ -(HMT $_{i,j}$) of skew shape (see Fig. 7(c)). A traditional task relocation mechanism [3] allows the submesh to slide up and become a new task reassignment (see Fig. 7(d)). Note that a simple sliding mechanism does not often result in the task reassignment having the form of a standard hierarchical 2D mesh tableau. Rather, a descent pair may occur and become a generalized hierarchical 2D mesh tableau. Recall that the task relocation mechanism in Algorithm 7 results in the task reassignment represented by a hierarchical 2D mesh tableau of normal shape (see Fig. 7(e)). For example, let A_0 be an initial task assignment represented by λ -(HMT $_{i,j}$) of skew shape in Fig. 7(c). Since A_0 has four empty processors in the top-left corner of λ -(HMT $_{i,j}$), the task reassignment sequence is $(A_0, A_1, A_2, A_3, A_4)$ by Algorithm 7. In the process of task reassignments, the greedy task relocation policy keeps a hierarchical 2D mesh tableau from occurring any descent pair. It follows that the resulting task reassignment A_4 in Fig. 7(e) is a necessarily optimal task reassignment for A_0 in Fig. 7(c).

7. Conclusions

In this paper we have presented a novel approach to representing priority-based task reassignments in a hierarchical 2D mesh-connected system (2D-HMS). A priority-based task reassignment in a 2D-HMS is represented by a hierarchical 2D mesh tableau λ -(HMT $_{i,j}$) in which task relocations under the greedy task relocation policy are reduced to a jeu de taquin slide on λ -(HMT $_{i,j}$). In case we ignore task relocation costs, the hierarchical nature of λ -(HMT $_{i,j}$) allows each task relocation to improve the task turnaround time. Some comparisons are given between the traditional approach and our approach to task relocations in a 2D-HMS. For a priority-based task assignment represented by λ -(HMT $_{i,j}$), a traditional approach may generate a descent pair for task relocations, which means that it is not a necessarily optimal task reassignment. We have shown that our approach to task reassignment keeps λ -(HMT $_{i,j}$) from occurring any descent pair for task relocations.

To the best of our knowledge, this paper is the first attempt to study task assignments and reassignments in 2D mesh-connected systems by using tableaux. However, the assumptions that we have made about priority-based task reassignments in a 2D-HMS still need to be relaxed. For example, we assumed that communications and task relocations are cost-free in a 2D-HMS. Priority-based task reassignments in a 2D-HMS with more relaxed assumptions (e.g., non-zero task relocation costs) are left for future work.

References

- [1] S. Ramakrishnan, I.-H. Cho, L. A. Dunning, A Close Look at Task Assignment in Distributed Systems, in: INFOCOM '91, IEEE, 1991, pp. 806 – 812.
- [2] A. S. Tanenbaum, Distributed Operating Systems, Prentice Hall, 1995.
- [3] S.-M. Yoo, H. Choo, H. Y. Youn, C. Yu, Y. Lee, On task relocation in two-dimensional meshes, Journal of Parallel and Distributed Computing 60 (5) (2000) 616–638.
- [4] B. S. Yoo, C. R. Das, A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers, IEEE Transactions on Computers 51 (1) (2002) 46–60.

- [5] U.-R. Chen, C.-C. Wu, W. Lin, Meshlization of Irregular Grid Resource Topologies by Heuristic Square-Packing Methods, *International Journal of Grid and Distributed Computing* 2 (4) (2009) 9–16.
- [6] X. Shen, W. Liang, Q. Hu, On Embedding Between 2D Meshes of the Same Size, *IEEE Transactions on Computers* 46 (1997) 880–889.
- [7] V. Nollet, P. Avasare, J.-Y. Mignolet, D. Verkest, Low Cost Task Migration Initiation in a Heterogeneous MP–SoC, in: *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, 2005, pp. 252–253.
- [8] P. K. Yadav, M. P. Singh, H. Kumar, Scheduling algorithm: tasks scheduling algorithm for multiple processors with dynamic reassignment, *Journal of Computer Systems, Networks, and Communications* 2008 (2008) 1–9.
- [9] M. Kafil, I. Ahmad, Optimal Task Assignment in Heterogeneous Distributed Computing Systems, *IEEE Concurrency* 6 (3) (1998) 42–51.
- [10] C. C. Shen, W. H. Tsai, A Graph Matching Approach to Optimal Task Assignment in Distributed Computing System Using a Minimax Criterion, *IEEE Trans. Computers* C-34 (1985) 197–203.
- [11] L.-L. Wang, Optimal assignment of task modules with precedence for distributed processing by graph matching and state-space search, *BIT* 28 (1) (1988) 54–68.
- [12] P. Manneback, E. M. Daoudi, J. Qin, Optimal Scheduling and Granularity for a 2D-grid precedence graph on a MIMD computer, in: *EUROSIM*, 1994, pp. 879–886.
- [13] S. H. Bokhari, Dual Processor Scheduling with Dynamic Reassignment, *IEEE Transactions on Software Engineering* 5 (1979) 341–349.
- [14] S.-Y. Cho, K. H. Park, Dynamic task assignment in heterogeneous linear array networks for metacomputing, in: *Proceedings of the Heterogeneous Computing Workshop '94*, 1994, pp. 66–71.
- [15] W. W. Chu, L. J. Holloway, M.-T. Lan, K. Efe, Task Allocation in Distributed Data Processing, *Computer* 13 (11) (1980) 57–69.

- [16] K. Efe, Heuristic models of task assignment scheduling in distributed systems, *Computer* 15 (1982) 50–56.
- [17] A. Mehrabi, S. Mehrabi, A. D. Mehrabi, An Adaptive Genetic Algorithm for Multiprocessor Task Assignment Problem with Limited Memory, in: *WCECS 2009*, 2009, pp. 1018–1023.
- [18] D. Kim, Representations of task assignments in distributed systems using young tableaux and symmetric groups, *arXiv.org arXiv:1012.1288 [cs.DC]*.
URL <http://arxiv.org/abs/1012.1288v3>
- [19] B. E. Sagan, *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*, 2nd Edition, Springer-Verlag, 2001.
- [20] W. Fulton, *Young Tableaux: With application to Representation Theory and Geometry*, Cambridge University Press, 1997.
- [21] R. P. Stanley, *Enumerative Combinatorics*, Vol. 2, Cambridge University Press, 1997.
- [22] Y. Zhao, Young tableaux and the representations of the symmetric group, *Harvard College Math Review* 2 (2) (2008) 33–45.
- [23] Z. Shi, E. Jeannot, J. J. Dongarra, Robust task scheduling in non-deterministic heterogeneous computing systems, in: *Proceedings of IEEE International Conference on Cluster Computing*, 2006, pp. 1–10.
- [24] O. Sinnén, *Task Scheduling for Parallel Systems*, Wiley-Interscience, 2007.
- [25] F. Suter, F. Desprez, H. Casanova, From Heterogeneous Task Scheduling to Heterogeneous Mixed Parallel Scheduling, in: *Euro-Par 2004 Parallel Processing*, 2004, pp. 230–237.
- [26] H. El-Rewini, T. G. Lewis, H. H. Ali, *Task scheduling in parallel and distributed systems*, Prentice-Hall, Inc., 1994.

- [27] K.-H. Seo, Fragmentation-Efficient Node Allocation Algorithm in 2D Mesh-Connected Systems, in: ISPAN '05: Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks, 2005, pp. 318–323.
- [28] I. Ababneh, An efficient free-list submesh allocation scheme for two-dimensional mesh-connected multicomputers, *The Journal of Systems and Software* 79 (8) (2006) 1168–1179.
- [29] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, *ACM Computing Surveys* 31 (4) (1999) 406–471.
- [30] D. E. Knuth, Permutations, matrices and generalized Young tableaux, *Pacific J. Math* 34 (1970) 709–727.
- [31] I. G. Macdonald, *Symmetric functions and Hall polynomials*, Oxford University Press, 1979.
- [32] R. Vessenes, Generalized Foulkes' Conjecture and tableaux construction, *Journal of algebra* 277 (2004) 579–614.
- [33] J. S. Frame, G. de B. Robinson, R. M. Thrall, The hook graphs of the symmetric group, *Canadian Journal of Mathematics* 6 (1954) 316–324.
- [34] J. B. Fraleigh, *A First Course in Abstract Algebra*, Addison-Wesley, Reading, 1998.
- [35] T. Hungerford, *Algebra*, Springer-Verlag, 1980.
- [36] G. de B. Robinson, On representations of the symmetric group, *Amer. J. Math.* 60 (1938) 745–760.
- [37] M. P. Schützenberger, Quelques remarques sur une construction de Schensted, *Math. Scand.* 12 (1963) 117–128.
- [38] B. Bollobás, *Modern Graph Theory*, Springer, 1998.
- [39] N. Biggs, *Algebraic Graph Theory*, Cambridge University Press, 1974.
- [40] S.-M. Lee, Every Young Tableau Graph Is d-Graceful, *Annals of the New York Academy of Sciences* 555 (1989) 296–302.

- [41] A. D. Kalvin, Y. L. Varol, On the generation of all topological sortings, *Journal of Algorithms* 4 (2) (1983) 150–162.
- [42] D. E. Knuth, *The art of computer programming. Vol. 3: Sorting and searching*, Addison-Wesley Publishing Company, 1973.
- [43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press, 2001.